BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images,
Please do not report the images to the
Image Problem Mailbox.

WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences | Cases

Search Results -

Term	Documents
(5 AND 4 AND 1 AND 6 AND 7).USPT.	2
(L6 AND L1 AND L4 AND L5 AND L7).USPT.	2

Database: US Patents Full-Text Database; US Pre-Grant Publication Full-Text Database; JPO Abstracts Database; EPO Abstracts Database; Derwent World Patents Index; IBM Technical Disclosure Bulletins

Search: L9

Refine Search | Recall Text | Clear

Search History

DATE: Thursday, June 12, 2003 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L9	l6 and l1 and l4 and l5 and l7	2	L9
L8	l6.ab. and l1 and l4 and l5	2	L8
L7	command near3 user$1	18891	L7
L6	resource$ near3 manag$	7026	L6
L5	quantit$ near2 resource$1	316	L5
L4	resource near2 capacity	580	L4
L3	request near2 user	17771	L3
L2	resource capacitY	189	L2
L1	resource near1 type	1324	L1

END OF SEARCH HISTORY

L9: Entry 1 of 2 File: USPT Dec 28, 1993

DOCUMENT-IDENTIFIER: US 5274813 A
TITLE: Operation system having a migration function which moves saved data associated with an interrupted process to a different save area

Abstract Text (1):
A computer system having a CPU which executes instructions and uses resources of the computer system as the instructions are being executed. A monitoring device of the computer system monitors the quantity of resources used by the execution device and outputs a signal indicating that a limiting threshold of the use of the computer system resources has been approached, reached, or exceeded. Once the system resource limiting threshold has been reached, the executing process in the CPU is interrupted and data associated with the executing program is saved in a first save area. If a user of the computer system does not take any action after a predetermined period of time, the saved data associated with the executing program is moved to a second save area which has a slower access time than the first save area. After another predetermined period of time has elapsed without input from the user indicating a resume of the previously executing program, the saved data is further moved to a different storage area having a slower access time than the second storage area. The saved data is used to restart the execution of the instructions which have been interrupted, upon the input of a restart command from a user, so that the computer does not have to re-execute the previously executed instructions.

Brief Summary Text (7):
In contrast to the program languages, since an OS is a system software group unique to each type of computer, it must provide a procedure system for compiling, linking and executing user programs (i.e. user definition programs) coded in a high-level language, and a multi-process environment (e.g. real-time multi-task processing system) in a normal computer. Therefore, the OS has a control system for performing management and scheduling of individual jobs, and management and monitoring of resources, and providing a user environment (e.g., utilities, etc.), and performs all the tasks necessary for operating the computer.

Brief Summary Text (9):
Important tasks to be executed by an OS include "management" and "monitoring" of resources. In general, the "resources" represent hardware (mainly, memory units, e.g., a main memory area, a semiconductor memory device, a magnetic memory device, etc. as external memory devices) which can be used by a job, and system software (e.g., a compiler for compiling a program coded in a high-level language, a linker, a loader, various utilities provided by a system, general-purpose software programs, and the like).

Brief Summary Text (13):
It is difficult to estimate not only the "calculation time (i.e. CPU occupation time)" of resources but also the "capacity" of a magnetic memory device. More specifically, in order to permanently store data in most of existing large-scale computers, after a data file is created on an external magnetic memory device, a memory area assigned to this file is utilized. When a user uses this magnetic memory device, he or she must designate a data capacity (e.g. size (MB)) in advance. In general, it is considered that the maximum capacity of a magnetic memory device is smaller than a physical memory space capacity, and is equal to the capacity of a logical memory space. When a user roughly estimates a data amount, he or she can calculate it based on the number of words, the word length, and the like. However,

in particular, when the user is a beginner, he or she often fails to correctly estimate the data amount. As a result, even though actual calculation processing is completely ended, in "write processing" executed after the calculation processing is ended, i.e., when data are finally written in a logical file of the magnetic memory device in the magnetic memory device of the set capacity, a series of processing operations may be abnormally terminated since the data cannot be completely written within the available capacity.

Brief Summary Text (22):
More specifically, according to the present invention, there is provided an OS for a computer, which compiles a program prepared by a user, and then executes a load module created by linking necessary modules, comprising means for monitoring an execution position of a user program, means for monitoring a value of a used resource of the computer, means for judging based on the monitoring results from these means whether the value (e.g., a required time or required capacity) of the used resource in use approaches a value a predetermined value to a designated upper limit value, or exceeds the upper limit value, means for, when the judging means judges that the value of the used resource approaches the value the predetermined value to the designated upper limit value exceeds the upper limit value, dumping contents of all registers and a status word register used to an external storage medium, temporarily terminating the processing (job), and informing to the user a message indicating that the resource is too small to execute the program, and means for, when the user performs a predetermined procedure, re-loading the already dumped contents, and allowing re-execution from an interrupted point, and more preferably, further comprising means for, when the user or an operator does not perform a predetermined procedure (e.g., does not change set values) within a predetermined term, automatically (i.e. without any user's interpose) deleting the dumped contents.

Detailed Description Text (37):
In the above description, the "CPU time" has been exemplified as a resource. However, substantially the same procedure is performed for other resources (e.g., a memory capacity, a memory space, etc.). A case will be briefly described below wherein the "memory capacity" is the resource.

Detailed Description Text (38):
When the memory capacity is a resource, the following two cases must be taken into consideration. That is, the first case is a case wherein the memory area is a "main memory" area, and the second case is case wherein the memory area is an "external memory" device. In the former case, since existing computers employ a "virtual memory system" except for some computers (in particular, super computers), the main memory area is basically independent from a physical memory (i.e. real memory space). However, a job which requires a wide main memory area beyond the performance of the virtual memory system having a virtual storage access method cannot be basically processed. In this case, the addressing method of the memory space must be drastically improved, and this means a change in overall architecture. Thus, such a case will be disregarded here.

Detailed Description Text (46):
Note that the present invention is not limited to the above embodiment. A resource is not limited to the CPU time, and an external memory device, and other resources may be used. As described in the above embodiment, some resources require proper pre-processing and post processing so as to re-execute a job from an interrupted point. The detailed procedure depends on the type of resource, and cannot be uniquely defined. However, such a procedure can be a modification of the present invention as long as an OS aims at re-executing a job from an interrupted point when a resource is exhausted, and performs an operation to prepare for the re-execution. The arrangement of the computer system is not limited to that shown in FIG. 1, and may be properly modified according to specifications. Various changes and modifications may be made without departing from the spirit and scope of the invention.

Other Reference Publication (1):
H. W. Lynch et al, The OS/VS 2 Release 2 System Resources Manager, vol. 13, No. 4, pp. 274-291 (1974).

CLAIMS:

1. An operating system for a computer, which compiles a program prepared by a user, then links predetermined modules to create a load module, and executes the load module as a job, which comprises:

program monitoring means for monitoring a present execution position of a statement of the user program and outputting program monitoring results;

used resource quantity monitoring means for monitoring quantities of used resources controlled by said operating system and outputting used resource quantity results; and

execution means for executing the user program and a control program for judging, based on the program monitoring results from said program monitoring means and the used resource quantity results output by the used resource quantity monitoring means, whether the quantity of a presently used resource approaches a quantity corresponding to a limiting threshold or exceeds the limiting threshold;

wherein when said execution means determines that the quantity of the used resource approaches the limiting threshold or has exceeded the limiting threshold, said operating system dumps the contents of the load module, all registers, and a status word register associated with the job of the user program to a predetermined external memory device, and temporarily interrupts the job processing;

said operating system further comprises means for, when it is determined that the quantity of the used resource approaches the quantity corresponding to the limiting threshold or has exceeded the limiting threshold, informing to the user a message indicating that "a resource is too small to execute the user program", and

when the user inputs a re-execution instruction from predetermined instruction input means, said operating system reloads the dumped contents to said execution means, and causes said execution means to re-execute the job from an interrupted point of the user program.

5. An operating system for a computer, which compiles a program prepared by a user, then links predetermined modules to create a load module, and executes the load module, which comprises:

program monitoring means for monitoring a present execution position of a statement of the user program and outputting program monitoring results;

used resource quantity monitoring means for monitoring quantities of used resources controlled by said operating system and outputting used resource quantity results; and

execution means for executing the user program and a control program for judging, based on the program monitoring results from said program monitoring means and the used resource quantity results output by the used resource quantity monitoring means, whether the quantity of a presently used resource approaches a quantity corresponding to a limiting threshold or exceeds the limiting threshold;

wherein when said execution means determines that the quantity of the used resource approaches the limiting threshold or has exceeded the limiting threshold, said operating system dumps the contents of the load module, all registers, and a status word register associated with the job of the user program to a predetermined external memory device, and temporarily interrupts the job processing,

said operating system further comprises means for, when said execution means judges that the quantity of the used resource approaches the quantity corresponding to the limiting threshold or has exceeded the limiting threshold, informing to the user a message indicating that "a resource is too small to execute the user program", and

when the user inputs a re-execution instruction, said operating system reloads the

dumped contents to said execution means, and causes said execution means to re-execute the job from an interrupted point of the user program,

wherein at least two external storage media are provided, and when the contents of the load module, all the registers, and the status word register are dumped, all of the contents are dumped to the storage medium having the shortest access time, and said operating system further comprises assignation means for, when no re-execution instruction according to the predetermined procedure is input after an elapse of a predetermined period of time, assigning the dumped contents to the storage medium having the second shortest storage medium, and for, when no re-execution instruction according to the predetermined procedure is input after an elapse of another predetermined period of time, sequentially migrating the dumped contents to the storage media having longer access times.

6. A method of controlling a computer by an operating system, comprising the steps of:

a used resource value comparison/judgment step of comparing a quantity of a resource used by a job with set quantities associated with resources which can be used and set in advance by a user when the job is normally executed, and judging whether a present quantity of the used resource has reached the set quantity;

an interruption pre-processing step of preparing for an interruption temporarily occurring in the job if the quantity of the used resource has reached the set value;

a dump processing step of dumping all contents of an executable program module, all registers, and a status word register to a first memory unit;

an information step of temporarily interrupting processing associated with the job by calling an interruption routine, and informing to a user a message indicating that the quantity of the resource is insufficient to complete the job;

a re-execution instruction determining step of determining if an instruction issued by the user is a re-execution instruction associated with the temporarily interrupted job processing;

a restart pre-processing step of executing pre-processing for restarting the interrupted job if the re-execution instruction determining step determines that the instruction issued by the user is the re-execution instruction;

a restart step of restarting the job in a normal execution state;

a first elapse time judging step of judging if a predetermined period of time has elapsed during which the contents are stored in said first memory unit without detecting the re-execution instruction from the user, and returning activation to the re-execution instruction determining step when the predetermined period of time has not elapsed;

an (N-1)th migration processing step of sequentially migrating the dumped contents to memory units having longer access times by migrating the content of an (N-1)th memory unit (N is an integer not less than 2) to an Nth memory unit when no re-execution instruction is issued within the predetermined period of time, and when a predetermined condition is satisfied thereafter;

an Nth elapse time judgment step of causing each Nth memory unit to check if the predetermined period of time has elapsed without detecting the re-execution instruction issued from the user, returning activation to the step immediately after the corresponding migration processing step when it is determined that the predetermined period of time has not elapsed, and executing the re-execution pre-processing step and the restart processing step in turn, and returning activation to normal execution when it is determined that the re-execution instruction is issued; and

a deleting step of deleting associated dumped files stored in all the memory units

used in all the migration processing steps in accordance with a predetermined procedure, an ending a series of task processing operations when no re-execution instruction is issued within the predetermined period of time.

16. A computer system, comprising:

execution means for executing computer system instructions of a program and using a resource of the computer system as the instructions are being executed;

monitoring means, connected to the execution means, for monitoring a quantity of the resource used by said execution means and outputting a monitoring result which indicates that the use of said resource has at least one of reached and exceeded a limiting threshold;

interrupt means, connected to the monitoring means and the execution means, for interrupting execution of said instructions when said monitoring means outputs the monitoring result which indicates that the use of said resource has at least one of reached and exceeded a limiting threshold;

save means, connected to the execution means, for saving data associated with the executing instructions in a first save area when said monitoring means outputs the monitoring result which indicates that the use of said resource has at least one of reached and exceeded a limiting threshold;

moving means, connected to the save means, for moving the data saved by said save means from the first save area to a second save area after an occurrence of a predetermined condition;

input means for inputting, after the execution of the instructions was interrupted, a command from a user which indicates that execution of said computer system instructions is to be restarted;

restart means, connected to the first and second save areas and the input means, for restarting execution of said computer system instructions by loading the saved data to a location in said computer system where said saved data is used to continue executing the interrupted computer instructions.

End of Result Set

L9: Entry 2 of 2 File: USPT Aug 3, 1993

DOCUMENT-IDENTIFIER: US 5233533 A
**** See image for Certificate of Correction ****
TITLE: Scheduling method and apparatus

Brief Summary Text (3):
Various scheduling methods exist to aid in and/or optimize such scheduling of resources. Typically, scheduling methods are based on one of two scheduling techniques, forward scheduling or backward scheduling. Briefly, in forward scheduling the sequence of desired manufacturing steps is scheduled beginning at the current date and from then into the future. The capacity of resources influences how far into the future steps are scheduled. Backward scheduling schedules the sequence of manufacturing steps in reverse order from the shipment date backward in time toward the current date without regard to the capacity of the resources required.

Brief Summary Text (5):
"Production Management Concept Cuts Delay, Budget Overruns" by J. C. Lowndes, Aviation Week and Space Technology, Volume 122, No. 19, Pages 85-87, May 13, 1985 discloses a scheduling software device called Opt (Optimized Production Technology). The Opt software combines capacity planning with forward scheduling and backward scheduling for "unconstrained" or infinite capacity resources. The core Opt module is a finite forward scheduler that uses resource capacity, orders and inventory data for optimum output from the resources. The core Opt module determines the hour and minute of each day at which bottlenecks or constrained processes will occur, and provides an indication of which due dates will be missed and by how much time the dates will be missed with the current plant configuration. A backward scheduling module of Opt assumes infinite capacity and schedules non-bottleneck processes beginning at the completion times determined by the core module. This ensures that material is available for the first operation in the critical part of the manufacturing process.

Brief Summary Text (7):
In accordance with the present invention a scheduling apparatus and method is provided which is based on finite forward and backward scheduling and which takes into consideration the capacity of resources. Dynamic selection of resources during routing of manufacturing processes is provided to minimize waste material. Also a scheduling apparatus and method is provided which calculates when materials are going to be needed and, hence, in what amount a manufacturer should buy materials.

Brief Summary Text (8):
Specifically, the present invention provides a scheduling software device with a supporting software program which employs a combination of backward and forward scheduling methods to schedule the sequence of events required to manufacture an item. In a sequence of events, A, B, C which must be performed by a target due date, the software program starts at the due date and looks backward in time for available required resources to accomplish event C. Once a resource with the needed availability is found for event C over a range of dates prior to the due date, the software program allocates the resource to event C for the necessary time period out of the range of dates. From the earliest allocated date backward in time, the software program schedules a resource to accomplish event B and so on such that events A, B and C are serially scheduled. If the necessary time to accomplish an event, (say, for example, event A) is longer than the time between the earliest

allocated day of a succeeding event (event B) and the present date, then the software program unschedules the previously allocated (backward scheduled) days and forward schedules resources from the present date into the future for events A-C. The forward scheduling also takes into account the capacity of the resources. The foregoing method is referred to as backward/forward scheduling.

Brief Summary Text (10):
In either of the foregoing scheduling methods, the software program takes into account the capacity of each machine and staffing pool of the plant. To that end, each manufacturing resource is assigned a manageable amount of work and is not overloaded or undersupplied with assignments. In a like manner, the inventory of materials is tracked and maintained.

Detailed Description Text (3):
The keyboard 11 and display unit 13 are coupled as I/O (input/output) devices to the digital processor 15 by means common in the art. That is, the keyboard 11 and/or a mouse (not shown) and the like are coupled to the processor 15 to provide the user a means for inputting data and commands to the digital processor 15. Display unit 13 is coupled to digital processor 15 to exhibit output of the digital processor. In addition, a printer (not shown) may be coupled to the workstation in a manner common in the art to provide hard copies of processor output.

Detailed Description Text (9):
The main procedure 89 schedules manufacturing steps of an item with respect to capacity of the resources required where manufacturing steps which have been scheduled for previously requested items use the same resources. Overall capacity of a resource is defined in memory area 19 and availability on any given date/time of a resource is indicated in allocation blocks 69, 71, 73 (FIG. 1a). Hence, the present invention software program provides a finite capacity scheduler.

Detailed Description Text (16):
The memory area 17 holding the descriptions of possible items is predefined through two screen views 21, 23 illustrated in FIGS. 2a and 2b respectively. Upon user command (i.e. by menu selection or operation of a function key or the like), digital processor 15 displays product list screen view 21 on display unit 13. The product list screen view 21 provides fields for the user to insert pertinent information to describe and specify an item (product). Through keyboard 11, the user provides the pertinent information in the respective fields of the screen view 21 while it is displayed on display unit 13.

Detailed Description Text (22):
A process ID is formed by the concatination of the process type and resource group. The digital processor 15 displays the process ID in the "Process ID Site Type" field of screen view 23. The user also indicates "in field "Process ID Site Type" the site (plant) associated with the process.

Detailed Description Text (53):
After the program of the present invention has arranged the Btree 83 of to-be-scheduled orders in least slack order, then the main procedure 89 (on user command) sequentially traverses the Btree and schedules each entry (order) in the Btree. During the scheduling of a user specified order, materials or other ingredients may be determined to be wanting as of a particular date. In order to satisfy the proper scheduling of the user specified order, the main procedure 89 places orders for the wanted materials. These orders are referred to as spawned orders. The main procedure 89 places the spawned order in a similar manner that the user specifies an "original" or parent order, i.e. by providing the pertinent item identification, site, quantity and date specification information to order file 67. Upon receipt of the spawned order from main procedure 89, digital processor 15 forms a record 99 for the spawned order in the order file 67 as described previously and illustrated in FIG. 1d.

Detailed Description Text (59):
To backward-forward schedule the series of processes, the main procedure 89 serially schedules in backward order the sequence of processes of an ordered item, from the specified required completion date of the order backward in time toward the first

day for scheduling. For each process, main procedure 89 determines and calls a pertinent resource manager (described later) to match the needs of the process with available resources as indicated by allocation blocks 69, 71, 73 (FIG. 1a) in processor memory (described in detail later). The allocation blocks indicate amounts of each resource already scheduled for use in manufacturing other items. In particular, the resource manager of a process determines available dates and times of resources that satisfy the process. If the sequence of the processes of the order cannot be successfully serially backward scheduled between the required completion date and the user specified first day for scheduling, the determined available dates and times of resources to satisfy processes are released and the sequence of processes for the order is forward scheduled from the first day into the future. During forward scheduling, the resource managers determine available dates and times of resources to satisfy the processes.

Detailed Description Text (62):
In serial backward order, beginning with the last process in the formed sequence of processes in process file 77 (FIG. 1d), for the order being scheduled, the main procedure 89 determines and calls a pertinent resource manager for the process. From a pre-established table cross referencing process numbers combined with process site to resource managers, the main procedure 89 determines the resource manager according to the process number and the process site of the corresponding process record in process file 77. Where a process employs more than one resource, main procedure 89 calls in parallel a different manager for each different resource for that process.

Detailed Description Text (63):
The called managers allocate respective resources in parallel. If the order is being backwards scheduled, the requested end date for each manager allocation is the process requested end date. If the order is being forward scheduled, the requested start date for each manager allocation is the process requested start date.

Detailed Description Text (64):
Hence, each manager of a particular process schedules its respective resources in parallel as to time and different managers for different processes of the ordered item schedule respective resources in series in backward order during backward scheduling and in sequence order during forward scheduling. Either direction of scheduling is done with regard to capacity of the involved resources.

Detailed Description Text (65):
Each manager of a process then makes a determination of how to satisfy the process where more than one resource is available for that process. As shown at 95 in FIG. 5 according to the manager name and plant site associated with that manager, a resource manager has a set of rules for determining which resources to use. The rules are organized according to a rule identification number. Associated with the rule identification number is a description of the rule and threshold (upper and lower limits) for which the rule is to apply. Also for each rule there is an indication of preference for using the rule. The preference is associated with a particular type of resource, for example the workstation type indicated in workstation file 31 (FIG. 1c) for a particular workstation. Also with the preference is indicated a capacity code for that type of resource. To that end, the preference provides minimization of waste of a resource. Also indicated is the number of days the user is willing to be early or late in order to use this preference.

Detailed Description Text (66):
Based on the rules the resource manager uniquely identifies a resource by its type, preference number and capacity code, as illustrated at 97 in FIG. 5. With these three identification factors the resource manager determines the particular resource to satisfy the subject process. From the record in the workstation parameter file 31 corresponding to the uniquely identified resource, the resource manager looks up the set up time for that resource, the run time for that resource, and the clear time for that resource. From these three time values, the manager determines the length of time for which the resource would be allocated to the process. Using the allocation blocks in memory of processor 15, the manager determines whether the resource to the process may be allocated for the determined length of time on the desired date/time being scheduled.

Detailed Description Text (67):
In the preferred embodiment, as illustrated in FIG. 1e for each resource there is an allocation block 69, 71 or 73. For a workstation/machine the resource allocation block 69 for each day indicates the three different shifts of that day. To allocate an amount of time of use of the workstation during a particular shift of a day, the resource manager creates an entry 20 under the desired shift. The entry 20 provides an indication of the particular block of time being allocated to the subject process. The entries 20 under each shift are maintained in chronological order for ease in searching availability of the workstation at desired dates and times.

Detailed Description Text (70):
The manager checks each resource according to the foregoing steps until a resource which is best suited for the desired date/time in the backward-forward scheduling scheme is found. The manager allocates that resource to the process by forming the pertinent entry to the allocation block for the resource. If the manager is unable to find a resource available at the desired date/time, the manager determines the best possible time when a suitable resource is available and returns this information to main procedure 89.

Detailed Description Text (72):
For purpose of further illustration and not limitation, the foregoing operations of resource managers is described next with reference to a preferred embodiment with three resource managers for three types of resources (materials, staff pool, and workstation/machines). A workstation manager selects a rule based on the batch size of the order. If a rule is not found for the batch size, the workstation manager returns to the main procedure 89 with an error flag. If a rule is found however, the manager loops through the preferences for that rule. For each preference the manager updates the capacity description and the capacity code for the preference. Then the workstation manager retrieves the workstation type defined by the workstation ID in the preference and tries to allocate the workstation/machine using the allocation block 69 (FIG. 1e) for the workstation. If the initial attempt to allocate fails (i.e., no available time is found during the desired shift) then the workstation manager proceeds to the next preference.

Detailed Description Text (77):
The materials manager is indexed by name and plant site. For each named manager, an alternate material manager is provided. Indexed by alternative manager identification is a list of material options. For each option, the list includes the material inventory name, plant site associated with the material, maximum percentage of material that plant site can contribute in order to satisfy the alternative and a minimum percentage. There is one materials manager for each material available as a resource in the manufacturing plant.

Detailed Description Text (78):
Main procedure 89 determines a materials manager according to process number and process site of the subject process. The determined materials manager searches the list of material options by material manager name, plant site of the manager and alternative identification number. As a result the materials manager uniquely identifies the material and percentage amounts of that material to satisfy the subject process. The materials manager then searches the material resource allocation block 73 (FIG. 1e) of the subject material for the desired date on which the material is to be allocated for use. Next the materials manager performs the following calculations. The manager sums all "daily net available amounts" from the current date up to the desired date on which the material is to be allocated for use. Each "daily net available amount" equals the amount of the material put into inventory that day minus the amount taken out of inventory that day. A second sum of the "daily net available amounts" from the day in question (i.e. the day on which the material is desired to be used) through all days scheduled into the future for that material is computed. The second sum is subtracted from the first sum to provide a resulting available quantity. If that resulting available quantity is greater than or equal to the quantity being requested for allocation then the materials manager schedules the amount being requested in the allocation block for the day in question.

Detailed Description Text (82):
After the main procedure 89 has completed scheduling all orders (parent and spawned) the allocation blocks 69, 71, 73 provide a indication of the various times and amount for which resources have been allocated. From the information held in these blocks various output maybe generated. In particular, an ordering plan may be generated for each order scheduled. The plan provides an indication of the processes involved to satisfy the order, the routing of the processes the types of resources used to satisfy each process, the start and end times of the resources used, the quantity of each resource used, and total elapsed time to satisfy the order.

Detailed Description Text (84):
Also various statistics on usage and the cost of holding inventory may be determined from the information held in the allocation blocks 69, 71 and 73. In one embodiment of the present invention, a database holds program output regarding the utilization of resources of the manufacturing plant. This database reports, according to the scheduled manufacturing processes, the quantity of use that each resource is scheduled to have. Resources represented in this database include workstations, staff pool and materials. As for workstations, an indication on a daily basis of what each workstation is scheduled to do is indicated. Also, over a period of time a percentage usage of days available is indicated. As for the staff pool resource, each staff pool group's daily work load is indicated and from the daily work load indication percentage of work load over a period of time is calculated and provided. As for the materials, the database indicates inventory levels on a day-to-day basis as well as over a period of predetermined time. From this database, bottleneck situations can be determined where 100% utilization is indicated. To that end, a manufacturer is able to determine whether it is necessary to add shifts or more staff or more workstations in his plant.

Detailed Description Text (87):
For example, main procedure 89 may schedule a sequence of processes to manufacture an item by a backward-jump forward method. In that method, serial backward scheduling of the sequence of processes is attempted as described previously. If any process in the sequence cannot be scheduled between the first day for scheduling and the required completion date, the main procedure 89 determines the number of days previous to the first day for scheduling that the processes at issue are able to be scheduled. This is accomplished by a temporary serial backward scheduling of the sequence of processes without regard to the first day for scheduling but with regard to capacity of involved resources as described previously. Upon completion of the temporary serial backward scheduling, the resulting scheduled begin date for manufacturing the subject item is subtracted from the first day for scheduling. From the subtraction, a resulting number of days previous to the first day for scheduling is obtained and held as a delta value. Main procedure deletes the temporary backward scheduling and adds the delta value to the required completion date to form an adjusted completion date. Main procedure 89 then backward schedules from the adjusted completion date by the previously described method with resource managers, preference rules, etc. If resources still are not found, a new delta value is calculated and a new adjusted completion date is established in the manner described above, and main procedure 89 attempts backward scheduling again.

L8: Entry 5 of 14 File: USPT Jun 13, 2000

DOCUMENT-IDENTIFIER: US 6075939 A
TITLE: Tightly coupled, scalable module based micro-kernel operating system architecture

Brief Summary Text (6):
The desired features of a modern operating system can be readily enumerated: stable, configurable for divergent computing environments, having minimal hardware requirements, low system management needs, high performance, if not providing near to hard real-time performance, low operating system execution overhead, support for third party kernel level software objects, adaptability to specialized computing environment applications, etc. The number of desired features seems to preclude the development of any operating system that is of general applicability over any significant range of computing system scale or purpose.

Brief Summary Text (8):
General purpose operating systems are considered to be less than adequately reliable for near and hard real-time applications. Although the Unix.RTM. operating systems are relatively mature and stable, their substantial kernel size and generality precludes use in most dedicated computer systems of moderate scale and certainly in small scale embedded system controller applications. That is, the Unix.RTM. operating system is regarded generally to be too large and monolithic to permit use in anything less than a large scale computer system. Further, many Unix.RTM. API system calls and low level kernel functions are not compatible with

Brief Summary Text (13):
While the micro-kernel/server design provides a substantial degree of modularity and extensibility, such operating systems tend to be inefficient in their use of available computing resources and in managing event responsivity. For example, a conventional micro-kernel/server design communicates by direct messages passed between the micro-kernel and the different then executing servers. Since each server executes in its own kernel thread, each message sent incurs at least two distinct context changes: one by the micro-kernel to forward a message and another by the server to perform the function requested by the message. Although kernel thread context switches are lightweight, all micro-kernel/server intercommunication is performed using messages. Further, blocking kernel functions, needed to maintain the data integrity of certain server functions, introduces unpredictable and unbounded event response latencies. Consequently, the micro-kernel/server operating system design is not well suited for high performance computing environments, particularly including those that require some bounded event response capabilities.

Brief Summary Text (14):
Consequently, there is a clear need for an efficient operating system architecture that is readily scalable, efficient in terms of execution performance and responsiveness events readily re-configurable to accept support for add or modify operating system services provided to application programs.

Brief Summary Text (21):
Yet another advantage of the present invention is that kernel components are provided to service sets of API system calls, permitting the supported API to be modified for specific operating system applications. Kernel components, particularly including those developed to serve a specialized purpose, can inclusively redefine and exclusively override existing API system calls, as well as adding new system calls, to support specialized functions and requirements of user/OEM applications,

all through a well understood, consistent operating system API model. The execution time overhead incurred in supporting the modifiable API structure is minimized. Also, an initialization time system of assigning system call handles ensures that a consistent, non-conflicting set of API system call identifiers is defined for every selected set of kernel components, including those developed by independent user/OEMs.

Brief Summary Text (22):
Still another advantage of the present invention is that, through selection of kernel plug-in components, operating systems of substantially different function, scale, and specialization can be realized from the operating system structure of the present invention. Very small embedded operating systems, moderate sized operating systems providing concurrent execution support for multiple general and special purpose applications, and large scale, fully featured, standards compliant operating systems capable of supporting multiple users and concurrently executing general and special purpose user applications.

Brief Summary Text (23):
A yet further advantage of the present invention is that, through selection of the included kernel components, the resource requirements of the operating system can be tailored to the available hardware resources available.

Detailed Description Text (12):
1) An initialization routine to establish the proper hardware system state appropriate for the start of the operating system boot process.

Detailed Description Text (16):
4) Several system time support routines including: a system clock initialization and clock interrupt routine; an interface to the hardware time clock; and a routine implementing a system timeout mechanism.

Detailed Description Text (20):
The kernel components 16-22 are each capable of implementing a wide variety of different kernel functions. Preferably, the function of kernel components can be generally categorized as sources of general applicability, services to support specific or special applications, and services to support a full scale configuration of the operating system 10. The following list of kernel functions represents a preferred set of kernel components.

Detailed Description Text (22):
Small to medium-scale operating system implementations can be efficiently constructed utilizing the embedded POSIX kernel component. This component implements a generally balanced set of kernel functions Additional functionality can be added by including any of the optional kernel components. These medium-scale systems are often used in technology areas including voice mail control and management applications, as the control system for moderate performance multi-function copier systems, etc.

Detailed Description Text (23):
Full large-scale operating systems can be created through the inclusion of both the POSIX and POSIX+ kernel components. The resulting functionality is appropriate for controlling general purpose computer systems both for general business use and more computationally intense scientific and engineering applications. Further specific capabilities can be added through the inclusion of any of the optional kernel components.

Detailed Description Text (24):
Fully Stand-Alone Operating System/Applications

Detailed Description Text (25):
In preferred embodiments of the present invention, the operating system 10 can be slightly restructured where only a single application program is to be supported by the operating system 10. The operating system is coupled directly to the application program, eliminating the need for an API system call interface Since only a single user level application is to be executed under the control of the operating system,

the entire resources of the operating system can be coupled directly to the application.

Detailed Description Text (26):
Although only a single application program can be supported under this slightly restructured architecture, there is no limitation on the use of the optional and alternate kernel components. Often, as is typical in embedded systems design, the effort is to minimize the size of the operating system and target application program. Consequently, there are few, if any, optional kernel components that provide sufficient functionality to outweigh the need for the truly minimum size embedded operating system and target application.

Detailed Description Text (27):
Regardless of whether additional kernel components are included, the restructuring permits the application program itself to become, in effect, a kernel component, reducing the execution overhead and resource requirements of supporting the API. The main routine of the application program effectively provides the main control loop that is executed by the operating system 10, once initialized.

Detailed Description Text (28):
Referring now to FIG. 2, the operating system 10 is shown in the larger context of a substantially complete computer system 30. The micro-kernel 12 is coupled with some number of kernel plug-in components, represented here by an I/O kernel component 32 and a file system kernel plug-in component 34. This portion of the operating system 10 is shown being largely supported by a generally standard computer hardware platform 36 that, in turn, connects to peripheral devices that present standard controls and return events to the standard hardware 64.

Detailed Description Text (29):
The portion of the operating system 10 represented by the micro-kernel 12 and the I/O and file system components 32, 34 combine to support a basic API 38 that is generally capable of supporting a user application 40. This user application 40 represents any number of different applications that are sufficiently supported through the API 38 to perform their intended purposes These user applications 40 may range from low speed event monitoring applications to complex and intensive hard real-time software development environments.

Detailed Description Text (30):
By way of example, a secured management kernel plug-in component 42 is shown as part of the operating system 10. This security management component 42 may implement proprietary encryption algorithms or simple access control and reporting functions. In either event, the proprietary security management component 42 presents an API extension 44 that, in combination with the API 38 serves to support a system monitoring application 46. Whether operating as a background system service or a high priority security monitoring and enforcement program, the system application 46 has full access to at least the portion of the operating system 10 represented by the micro-kernel 12 and kernel components 32, 34, 42.

Detailed Description Text (31):
Process control and real-time control components 48, 50 are presented as included kernel components within the operating system 10. These components 48, 50 are interfaced with custom computer hardware 52 and OEM hardware 54. The custom and OEM hardware may preferably provide control data and receive operating event notifications from peripheral devices represented as the special controls and events block 56.

Detailed Description Text (32):
As an example, a user of the computer system 30 has acquired OEM hardware 50 and real-time control kernel component 50 from a third party vendor of such components and hardware 50, 54. Additionally, the user of the computer system 30 has developed the process control component 48 and custom hardware 52 to provide independent or supplementary functions desired to support proprietary API extensions 58, 60. A custom application 62 may be developed in part by the OEM vendor of the control component 50 and hardware 54. Further development of the custom application 62 to handle some different or larger class of control processes, utilizes the API

extension 58 to access the process control component 48.

Detailed Description Text (33):
Similarly, the custom application 62 may further reference the API 38 to obtain standard system services available from the operating system 10 as represented by the kernel components 32, 34.

Detailed Description Text (51):
Subsequently, application programs 124, 126 can call the well known API system call entry point of the kernel component 112. This call is made with a name argument, the same name used by the kernel component 116, to retrieve the previously assigned API system call number now used by the kernel component 116.

Detailed Description Text (52):
Similarly, the kernel component 120 can register with the component 112, based on a unique name, to obtain a unique API system call number, and create the desired API system call point. The user application 126, can determine the API system call number assigned to the kernel component 120 by presenting the same name to the kernel component 112 and receiving back the API system call number being used by the kernel component 120.

Detailed Description Text (63):
The boot related system hardware is first initialized 132 to a known good state. A minimal memory management system is then established 134 for the short term use of system memory.

Detailed Description Text (65):
The available interrupt and event control hardware is configured and established 136 for initial use.

Other Reference Publication (2):
Kosai et al., "Application of Virtual Storage to Switching Systems--An Application of CTRON Kernel On a General-Purpose Microprocessor", NTT Electrical Communication Laboratories, IEEE, pp. 0465-0469, 1989.

Other Reference Publication (4):
Engler et al., "Exokernel: An Operating System Architecture for Application-Level Resource Management", ACM, pp. 251-266, Dec. 1995.

Other Reference Publication (7):
Keshav, "Proceedings of the conference on Communications Architectures, Protocols and Applications", ACM, pp. 149-157, 1994.

Other Reference Publication (8):
Gheith et al., "CHAOSarc: Kernel Support for Multiweight Objects Invocations, and Atomicity in Real-Time Multiprocessor Application", ACM, pp. 33-72, Feb. 1993.

L8: Entry 1 of 2 File: USPT Aug 28, 2001

DOCUMENT-IDENTIFIER: US 6282561 B1
TITLE: Method and system for resource management with independent real-time applications on a common set of machines

Abstract Text (1):
A resource management mechanism is provided to ensure that real-time application programs running on a single machine or set of machines exhibit predictable behavior. The resource management mechanism employs the abstraction of an activity which serves as the basis for granting resource reservations and for accounting. An activity submits a request for resources in specified amounts to a resource planner. The activity is resource self-aware so that it is aware of its resource requirements. The activity may query resource providers to obtain resource requirements for particular operations. The resource planner determines whether the activity should be granted the requested reservation by employing an internal policy. Policy is separated by mechanism so that the resource planner may implement any of a number of policies. The resource planner may choose to grant the reservation to an activity or deny the request by an activity. When denying a request, the resource planner may inform the activity of what quantity of the requested resources are currently available so that the activity may submit a modified request. The resource management mechanism includes a dynamic feedback mechanism for initiating renegotiation of resource reservations when appropriate.

Brief Summary Text (4):
Conventional resource management strategies for real-time application programs have been unsatisfactory. A "real-time application program" is an application program that must execute in a predictable and timely fashion in order to operate properly. Many current efforts to provide resource management for real-time application programs may only manage a static set of resources. In other words, the resource set may not change during the course of operation of the system. Another limitation of many conventional resource management strategies is that they only accommodate one type of resource (i.e., the resources must be homogeneous). An additional limitation of many conventional resource management strategies is that they rely upon the resource or the application program to determine which application program should be allocated a resource and what quantity of the resource should be allocated. Another conventional approach has been to rely upon human experts to correctly assign priorities to tasks within the system. Unfortunately, such human experts typically cannot accurately assign priorities for a dynamic mix of real-time application programs.

Detailed Description Text (2):
The preferred embodiment of the present invention provides a resource management mechanism for arbitrating resource requests and resource usage among independent real-time application programs that run simultaneously on a single machine or set of machines. The resource management mechanism is adaptable to installation on a distributed system having separate computer systems by installing the mechanism on each of the computer systems within the distributed system. The resource management mechanism utilizes dynamic feedback to adapt to changing resource availability and resource requirements. In addition, the resource management mechanism is extensible so that the resource management employed in the system accounts for new resources that are added to the system and for resources that are removed from the system. The resource management mechanism separates resource management policy from resource management mechanism such that the mechanism is independent of the policy. The

mechanism for implementing policy is generalized so as to facilitate any one of a number of different policies. In the preferred embodiment of the present invention, real-time application programs are resource self-aware in that they are aware of what resources they require and what quantity of those resources they require. The applications negotiate with the resource management mechanism to reserve resources and ensure predictable performance.

Detailed Description Text (6):
In general, an activity seeks to reserve resources that it needs before it uses those resources. An activity reserves resources by requesting the reservation of a number of resources from the resource planner. The activity specifies the quantity of each resource it wishes to reserve. The resource planner applies a policy to the request and determines whether the resources should be granted or not in view of pending reservations. If the resources are granted to the activity, the activity may proceed to use the resources. In such a case, the reservations are utilized in the scheduling of resources. The scheduling of the resources described in more detail in copending application Ser. No. 08/568,577, now U.S. Pat. No. 5,812,844 entitled "Method and System for Scheduling the Execution of Threads Using Optional Time-Specific Scheduling Constraints", which was filed or even date herewith and which is assigned to a common assignee with the present application. If the request is not granted, the activity should not use the resource because the activity cannot be assured that adequate amounts of the resource will be available for the activity to run predictably.

Detailed Description Text (9):
As was mentioned above, in the preferred embodiment of the present invention real-time application programs are resource self-aware. The application programs 20 and 22 know what resources, as well as how much of those resources, they need to have to run properly and predictably. FIG. 2 is a flowchart illustrating the steps that are performed when an activity that is associated with an application program seeks to reserve resources. Initially, the activity determines what resources it needs (step 34 in FIG. 2). This determination involves an information gathering process. In the preferred embodiment of the present invention, the activity queries resource providers to determine what resources the activity needs. The activity is aware of what resource providers it uses, and the activity queries the resource providers it uses to determine what resources are, in turn, needed by those resource providers to perform their job and what quantities of resources are required for the resource providers to perform their job.

Detailed Description Text (15):
Once the activity determines what resources it needs and the quantity of those resources it needs, the activity sends a request for the determined quantities of resources to local resource planner to obtain a reservation (step 36 in FIG. 2). The request that is submitted by the activity holds a "resource set." The resource set specifies what resources are requested and what quantity of those resources are requested. The resource set includes a number of pairs of resources and resource amounts. More formally, each pair of a resource set includes a reference to an IResource interface supported by a particular resource along with an amount specified in terms of the units of that resource. The IResource interface will be described in more detail below.

Detailed Description Text (18):
FIG. 5 shows an example in which an activity 60 sends a resource set to the resource planner 62 to request quantities of resources. The resource planner 62 receives the resource set and applies an appropriate policy to determine whether the resources should be granted (step 38 in FIG. 2). If the resources are granted, the activity 60 may use the reserved resources (step 40 in FIG. 2). If, on the other hand, the resources are not granted, the resource planner 62 informs the activity 60 of the quantities of the requested resources that are available, if any (step 42 in FIG. 2). The activity 60 then may determine whether the available resources are acceptable (step 44 in FIG. 2). If such available resources are acceptable, the activity 60 may reformulate its reservation request to conform to the current availability of the resources (step 46 in FIG. 2). The negotiation process then is repeated by continuing execution at step 38 in FIG. 2. If, however, the available resources are not acceptable, the activity terminates.

Detailed Description Text (21):
The Reserve( ) method allows the resource planner 62 to reserve a certain quantity of the resource. Hence, for the example shown in FIG. 5, the resource planner 62 makes calls 72A, 72B, 72C, 72D and 72E to call the Reserve( ) methods of the IResource interfaces supported by the resource providers 50, 52, 54, 56 and 58, respectively.

Detailed Description Text (22):
As can be seen above, the IResource interface includes a TotalAmount( ) method that allows the resource planner 62 to determine the total amount or capacity of the resource in the resource specific units. The GetReservation( ) method returns the amount of the resource that is reserved for a particular activity but is specified in the parameters for the call to this method. The IResource interface also includes a GetFree( ) method that returns a value specifying a current amount of the resource that is available. The GetUsage( ) method returns the actual usage of the resource by an activity.

Detailed Description Text (24):
The RequestResources( ) method is passed a resource set and passes out a value that specifies either that the resource reservation for the input resource set was granted or was not granted. If the request was not granted, a resource set is passed out that specifies the amount of available resources of the type asked for. The ReleaseResources( ) method releases all the resources that are reserved by an activity, and the CreateResourceSet( ) method creates a resource set.

Detailed Description Text (26):
Initially, the resource planner receives a request for resources from an activity (step 74 in FIG. 6A). The resource planner then checks whether the requested resources are currently available in the requested amounts (step 76 in FIG. 6A). If the resources are available in the requested amounts, the resources are granted to the activity (step 78 in FIG. 6A). If the resources are not all available in the requested quantities, the resource planner checks whether any lower importance activities are using resources that are requested so that the resources may be reassigned to complete the resource reservation of the requesting activity (step 80 in FIG. 6A). The policy of the preferred embodiment in the present invention employs the notion of importance where activities can be ranked according to importance, and the importance of activities may be compared. If lower importance activities are using sought resources that have been requested by the higher importance requesting activity, these resources are reassigned to be granted to the higher importance requesting activity in order to completely satisfy the resource reservation request (step 82 in FIG. 6A).

Detailed Description Text (28):
The resource planner calls the OnNeed( ) method to inform the lower importance activity of the quantity of resources that are reserved by that activity are needed by other activities. The lower importance activity then resubmits a new reservation request relinquishing the resources that are needed by the higher importance activity (step 77 in FIG. 6B).

WEST

End of Result Set

Generate Collection Print

L8: Entry 2 of 2 File: USPT Nov 29, 1994

DOCUMENT-IDENTIFIER: US 5369570 A
TITLE: Method and system for continuous integrated resource management

Abstract Text (1):
A method for continuous real-time management of heterogeneous interdependent resources is described. The method preferably comprises using multiple distributed resource engines to maintain timely and precise schedules, and action controls, and identifying and responding to rapidly changing conditions in accord with predetermined requirements, relationships, and constraints. Each resource engine continuously adjusts schedules in response to changing status, resource requirements, relationships and constraints. Each action control maintains an ordered list of conditions requiring action, determines the best action in each case, and generates appropriate responses. Preferably methods for continuous operation include inquiring about status concurrent with scheduling activity and recognizing the effects of time passage on the condition of schedules.

Brief Summary Text (24):
To provide a global view of decision alternatives, consistent information from strategic, tactical, and operational viewpoints are required. The present invention provides event information and accommodates diverse resource types, relationships, constraints, and multiple sets of rules for future intervals of time. Heterogeneous resources are preferably scheduled simultaneously in a common process to integrate the management of capacity and materials schedules at multiple levels of detail. A single integrated schedule can contain long term forecasts, actual customer orders, and production schedules, with a level of detail and time resolution appropriate to each. This eliminates the need for redundant information, eliminates inconsistency, and provides demand (up) and supply (down) visibility from strategic through operational management levels.

Drawing Description Text (4):
FIG. 3 is a table and graph illustrating the representation of supply and demand events for a material type resource.

Drawing Description Text (5):
FIG. 4 is a table and graph illustrating the representation of supply and demand events for a capacity "rate" resource.

Detailed Description Text (22):
These notices are acted upon automatically or through intervention of a decision process by one or more operators/users in the response 105. As described more fully herein, a decision process has three essential parts: discovery, choice, and action. Each operator is provided with a action list prioritized by rules defining the relative merit of each resource and type of condition. Alternative actions (choices) are provided, from which corrective command primitives are generated without data entry. These command primitives are then directed to one or more resource engines for processing as described in 102, 103, and 104. These and other functions support a timely and interactive decision process 107 & 108. The entire process above is repeated continuously.

Detailed Description Text (34):
Resource bills have failed best efforts to adequately describe the full spectrum of

resource types necessary for true integrated resource management. A multitude of scheduling problems often require custom solutions because the specific characteristics of resources being managed are atypical. A preferred embodiment uses common terminology, data, and process to schedule a multitude of resource types including, but not limited to, materials, machine and labor capacity, and time itself.

Detailed Description Text (39):
The example of FIG. 3 is a material type resource. Parameters describing the behavior of a resource type specify if amounts are treated as absolute values (material) or rates (capacity), and whether the balance is retained or reset at defined intervals. Values of time 303 & 304 and the amount 306 are the raw data, and event type in conjunction with resource type determines the behavior of the resulting balance 307. In this material example transitions in graph line 308 are vertical because the amount 306 is treated as an absolute value of change and the balance does not decay to zero because it is persistent.

Detailed Description Text (40):
FIG. 4 illustrates a schedule of events with rates (typically machines & workers) in TABLE 401, graph 407, and line 408. The TIME-1 402 is still the point in future time when the resource's amount first changes. The TIME-2 403 is greater than TIME-1 and determines when the event ends. Events with a duration (TIME-2 minus TIME-1) have an AMOUNT 405 that is a "rate". The AMOUNT 405 multiplied by the duration is the amount by which the resource is consumed or supplied. FIG. 4 is similar to FIG. 3 but is a capacity-type resource. The first event is supply event of beginning at TIME-1 402, ending at TIME-2 404, of amount 405. The difference between the two times is still the duration 409. TIME-1 is still the time when resource balance is impacted. TIME-2 is GREATER than TIME-1 indicating that the rate 409 is now represented by the amount 405. Event 1 has a rate of 10 as shown in 409, while event 3 has a higher rate of 40 as shown in 411.

Detailed Description Text (42):
A comparison of TIME-1 and TIME-2 identifies when an event expresses a rate or an absolute amount. The event and resource types allow for all possible combinations of positive and negative impacts and persistence in determining the balance. An minimum number of variables and parameters are used in conjunction with simple logic to produce schedules that accurately represent realistic schedules for a multitude of resources.

Detailed Description Text (50):
The resource engine preferred embodiment consists of consists of 11 component parts for which essential algorithms and data structures are described herein. The sequence and hierarchical relationships shown in FIG. 6 are different from ordinary methods. Special care was taken to develop new methods to provide the behavior required for continuous operation with concurrent inquiry and update while assuring fault tolerance or recoverability. Special care was also taken to completely integrate the data and processes of capacity and material resources, and to manage such information so that a single data set could be employed for strategic, tactical, and operational resource management. A preferred embodiment is an object oriented design whose methods manage data directly and whose inputs and outputs are in message form. Cooperative rather than multi-processing methods were used to achieve practical concurrent operation of component parts 4, 7, and 8. Data structures and algorithms were designed to achieve maximum performance on a general purpose digital computing apparatus.

Detailed Description Text (77):
(a) Characteristics 802 include the external (long form) resource identification (KEY), internal resource identification (RID), resource type, level value, user identification (UID), time when last updated, time of next trigger, current amount (Inventory), process yield factor, Units-Of-Measure (UOM), and of course a text description of the resource.

Detailed Description Text (80):
(d) Dependent resources data 805 defines all dependent requirements for material and capacity consistently. A capacity resource (machine) can define material (tool) or

capacity (maintenance) requirements, or visa versa, The representation of material and capacity events in schedules and determination of dependent demand event (requirement) amount is described herein. Only scheduled dependent resources are included in the resource database. History, product and process documentation, and all other data irrelevant to scheduling is eliminated. The units conversion component of the present invention compiles (pre-processes) dependent relationships, reducing the descriptive information and processing required at run-time. A relationship is defined by the dependent resource identifier (DRID), factor value, and calculation method code.

Detailed Description Text (116):
Exploding parent to dependent requirements is critical to high performance and correct consistent results. A preferred embodiment of the resource engine can process diverse resource types with potentially complex units conversion and rounding. The present invention contemplates infrequent relationship changes relative to determination of dependent amounts.

Detailed Description Text (163):
2. The total of all event quantities within the current interval 510 is computed (firm+open-requirements-allocation -forecast). When the duration is positive (a rate-type resource) quantity is multiplied by duration.

Detailed Description Text (165):
4. As each interval's processing is completed, the next interval's position is determined by calendar/rules. The current resource balance is either carried forward or reset depending upon the type of resource. Capacity-type resources set the interval's available resource to the amount of that resource available for the present supply interval (8 hours per shift). Materials-type resources (no duration) carry forward any previous amount (inventory).

Search Results

Search Results for: **[(resource) AND (type) AND (NOT capacity)]**
Found **19,815** of **127,944 searched.**

Warning: Maximum result set of 200 exceeded. Consider refining.

Search within Results

GO > Advanced Search

> Search Help/Tips

Sort by: **Title** **Publication** **Publication Date** **Score**

Results 1 - 20 of 200 **short listing**

Prev Page 1 2 3 4 5 6 7 8 9 10 Next Page

1 Abstracting remote object interaction in a peer-2-peer environment 100%

Patrick Thomas Eugster , Sebastien Baehni

Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande November 2002

Leveraged by the success of applications aiming at the "free" sharing of data in the Internet, the paradigm of peer-to-peer (P2P) computing has been devoted substantial consideration recently.This paper presents an abstraction for remote object interaction in a P2P environment, called borrow/lend (BL). We present the principles underlying our BL abstraction, and its implementation in Java. We contrast our abstraction with established abstractions for distributed programming such as the remote me ...

2 Resource &equil; abstract data type + synchronization - A methodology for message oriented programming - 100%

P. R.F. Cunha , T. S.E. Maibaum

Proceedings of the 5th international conference on Software engineering March 1981

We present in this paper a methodology for the development (and analysis) of programs designed specifically for distributed environments where synchronization is achieved through message passing. The methodology is based on techniques and concepts which have been found to be useful for the development of sequential programs—namely, stepwise refinement and abstract data types. The methodology is based on the concept of resource, generalizing the concepts of monitors, managers, propriet ...

3 Time regions and effects for resource usage analysis 100%

Naoki Kobayashi

ACM SIGPLAN N tices , Proceedings of the 2003 ACM SIGPLAN international worksh p n Types in languages design and implementation January 2003
Volume 38 Issue 3

Various resources such as files and memory are associated with certain protocols about how they should be accessed. For example, a memory cell that has been allocated should be eventually deallocated, and after the deallocation, the cell should no longer be accessed. Igarashi and Kobayashi recently proposed a general type-based method to check whether a program follows such resource access policies, but their analysis was not precise enough for certain programs. In this paper, we refine their ty ...

4 Query Language for Semantic Web: EDUTELLA: a P2P networking infrastructure based on RDF 100%

Wolfgang Nejdl , Boris Wolf , Changtao Qu , Stefan Decker , Michael Sintek , Ambjörn Naeve , Mikael Nilsson , Matthias Palmér , Tore Risch
Proceedings of the eleventh international conference on World Wide Web May 2002

Metadata for the World Wide Web is important, but metadata for Peer-to-Peer (P2P) networks is absolutely crucial. In this paper we discuss the open source project Edutella which builds upon metadata standards defined for the WWW and aims to provide an RDF-based metadata infrastructure for P2P applications, building on the recently announced JXTA Framework. We describe the goals and main services this infrastructure will provide and the architecture to connect Edutella Peers based on exchange of ...

5 Query Language for Semantic Web: RQL: a declarative query language for RDF 100%

Gregory Karvounarakis , Sofia Alexaki , Vassilis Christophides , Dimitris Plexousakis , Michel Scholl
Proceedings of the eleventh international conference on World Wide Web May 2002

Real-scale Semantic Web applications, such as Knowledge Portals and E-Marketplaces, require the management of large volumes of metadata, i.e., information describing the available Web content and services. Better knowledge about their meaning, usage, accessibility or quality will considerably facilitate an automated processing of Web resources. The Resource Description Framework (RDF) enables the creation and exchange of metadata as normal Web data. Although voluminous RDF descriptions are alrea ...

6 Resource usage analysis 100%

Atsushi Igarashi , Naoki Kobayashi
ACM SIGPLAN Notices , Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages January 2002
Volume 37 Issue 1

It is an important criterion of program correctness that a program accesses resources in a valid manner. For example, a memory region that has been allocated should be eventually deallocated, and after the deallocation, the region should no longer be accessed. A file that has been opened should be eventually closed. So far, most of the methods to analyze this kind of property have been proposed in rather specific contexts (like studies of memory management and verification of usage of lock primi ...

7 Trust and partial typing in open systems of mobile agents 100%

James Riely , Matthew Hennessy
Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages January 1999

8 Distributed deadlock detection in Ada run-time environments 100%

Chia-Shiang Shih , John A. Stankovic
Proceedings of the conference on TRI-ADA '90 December 1990

Distributed deadlock detection has been studied in distributed database systems and distributed timesharing operating systems, but has not been widely used in real-time systems such as Ada runtime environments. In this paper we are interested in explicitly tying the formal properties of deadlock algorithms to Ada and its runtime system. We analyze and categorize the deadlock problem in Ada environments into four levels of complexity by using Knapp's hierarchy of deadlock models. To fully su ...

9 The m-calculus: a higher-order distributed process calculus 100%

Alan Schmitt , Jean-Bernard Stefani
ACM SIGPLAN Notices , Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages January 2003
Volume 38 Issue 1

This paper presents a new distributed process calculus, called the M-calculus, that can be understood as a higher-order version of the Distributed Join calculus with programmable localities. The calculus retains the implementable character of the Distributed Join calculus while overcoming several important limitations: insufficient control over communication and mobility, absence of dynamic binding, and limited locality semantics. The calculus is equipped with a polymorphic type system that guar ...

10 Resource management scheme in distributed environments 100%

O. Nakamura , N. Saito
ACM SIGCOMM Computer Communication Review , Proceedings of the ACM workshop on Frontiers in computer communications technology August 1987
Volume 17 Issue 5

The user interface for distributed computing environment needs several kinds of transparency for resources which are distributed on many sites connected to the network. Access transparency and location transparency are general concept for distributed resources. In distributed environment, there are many resources of the same functions in various sites. For example, cashed resources which are complete duplicated resources exist on many sites in order to realize high performance.

11 Estimation of lower bounds in scheduling algorithms for high-level synthesis 100%

Giri Tiruvuri , Moon Chung
ACM Transactions on Design Automation of Electronic Systems (TODAES) April 1998
Volume 3 Issue 2

To produce efficient design, a high-level synthesis system should be able to analyze a variety of cost-performance tradeoffs. The system can use lower-bound performance estimated methods to identify and puune inferior designs without producint complete designs. We present a lower-bound performance estimate method that is not only faster than existing methods, but also produces better lower bounds. In most cases, the lower bound produced by our algorithm is tight.Scheduling algori ...

12 Workshop on software engineering decision support: processes: A policy-based resource instantiation mechanism to automate software process management 100%

Carla A. Lima Reis , Rodrigo Quites Reis , Heribert Schlebbe , Daltro J. Nunes

Proceedings of the 14th internati nal conference on Software engineering and knowledge engineering July 2002

Process-Centered Software Engineering Environments (PSEEs) deal with activities that demand specialized personnel and limited resources. Characteristics about required resources and people (and their dynamic availability) are used by software process instantiation phase to define process allocation strategies. However, most of existing PSEEs do not allow precise resource specification, and the instantiation is often based on the knowledge of a process designer, mostly without automated support. ...

13 Enabling knowledge representation on the Web by extending RDF schema 100%

Jeen Broekstra , Michel Klein , Stefan Decker , Dieter Fensel , Frank van Harmelen , Ian Horrocks

Proceedings of the tenth international conference on World Wide Web April 2001

14 An expressive, scalable type theory for certified code 100%

Karl Crary , Joseph C. Vanderwaart

ACM SIGPLAN Notices , Proceedings of the seventh ACM SIGPLAN international conference on Functional programming September 2002

Volume 37 Issue 9

We present the type theory LTT, intended to form a basis for typed target languages, providing an internal notion of logical proposition and proof. The inclusion of explicit proofs allows the type system to guarantee properties that would otherwise be incompatible with decidable type checking. LTT also provides linear facilities for tracking ephemeral properties that hold only for certain program states.Our type theory allows for re-use of typechecking software by casting a variety of type syste ...

15 Access control for mobile agents: The calculus of boxed ambients 100%

Michele Bugliesi , Giuseppe Castagna , Silvia Crafa

ACM Transactions on Programming Languages and Systems (TOPLAS) January 2004

Volume 26 Issue 1

Boxed Ambients are a variant of Mobile Ambients that result from dropping the open capability and introducing new primitives for ambient communication. The new model of communication is faithful to the principles of distribution and location-awareness of Mobile Ambients, and complements the constructs in and out for mobility with finer-grained mechanisms for ambient interaction. We introduce the new calculus, study the impact of the new mechanisms for communication of typing and mobility, ...

16 Software development control based on module interconnection 100%

Walter F. Tichy

Proceedings of the 4th international c nference n Software engineering September 1979

Constructing large software systems is not merely a matter of programming, but also a matter of communication and coordination. Problems arise because many people work on a joint project and use each other's programs. This paper presents an integrated development and maintenance system that provides a controlling

environment to insure the consistency of a software system at the module interconnection level. It assists the programmers with two facilities: In ...

17 Open hypermedia and the web: Xspect: bridging open hypermedia and XLink 100%

Bent G. Christensen , Frank Allan Hansen , Niels Olof Bouvin
Proceedings of the twelfth international conference on World Wide Web May 2003

This paper evaluates the XLink format in comparison with other linking formats. The comparison is based on Xspect, an implementation of XLink. Xspect handles transformation between an open hypermedia format (OHIF) and XLink, and the paper discusses this isomorphic transformation and generalises it to include another open hypermedia format, FOHM. The Xspect system, based on XSLT and Javascript, provides users with an interface to browse and merge linkbases. Xspect supports navigational hypermedia ...

18 The distributed deadlock detection algorithm 100%

D. Z. Badal
ACM Transactions on Computer Systems (TOCS) September 1986
Volume 4 Issue 4

We propose a distributed deadlock detection algorithm for distributed computer systems. We consider two types of resources, depending on whether the remote resource lock granularity and mode can or cannot be determined without access to the remote resource site. We present the algorithm, its performance analysis, and an informal argument about its correctness. The proposed algorithm has a hierarchical design intended to detect the most frequent deadlocks with maximum efficiency.

19 Selective memoization 100%

Umut A. A. Acar , Guy E. Blelloch , Robert Harper
ACM SIGPLAN Notices , Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages January 2003
Volume 38 Issue 1

We present a framework for applying memoization selectively. The framework provides programmer control over equality, space usage, and identification of precise dependences so that memoization can be applied according to the needs of an application. Two key properties of the framework are that it is efficient and yields programs whose performance can be analyzed using standard techniques.We describe the framework in the context of a functional language and an implementation as an SML library. Th ...

20 Enforcing high-level protocols in low-level software 100%

Robert DeLine , Manuel Fähndrich
ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation May 2001
Volume 36 Issue 5

h c g e cf

Inc.

h c g e cf

Search Results

Search Results for: **[(resource) AND (type and capacity)]**
Found **5,914** of **127,944 searched.**
Warning: Maximum result set of 200 exceeded. Consider refining.

Search within Results

GO > Advanced Search

> Search Help/Tips

Sort by: **Title** **Publication** **Publication Date** **Score**

Results 1 - 20 of 200 **short listing**

Prev Page 1 2 3 4 5 6 7 8 9 10 Next Page

1 Circuit partitioning with complex resource constraints in FPGAs 100%
Huiqun Liu , Kai Zhu , D. F. Wong
Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays March 1998
In this paper, we present an algorithm for circuit partitioning with complex resource constraints in large FPGAs. Traditional partitioning methods estimate the capacity of an FPGA device by counting the number of logic blocks, however this is not accurate with the increasing capacity and diverse resource types in the new FPGA architectures. We propose a network flow based method to optimally check whether a circuit or a sub-circuit is feasible for a set of available heterogeneous resources. ...

2 An inversion algorithm to compute blocking probabilities in loss networks with state-dependent rates 99%
Gagan L. Choudhury , Kin K. Leung , Ward Whitt
IEEE/ACM Transactions on Networking (TON) October 1995
Volume 3 Issue 5

3 Multiplexing gains in bit stream multiplexors 99%
Ikhlaq Sidhu , Scott Jordan
IEEE/ACM Transactions on Networking (TON) December 1995
Volume 3 Issue 6

4 Computer Communication Networks: Approaches, Objectives, and Performance Considerations 98%
Stephen R. Kimbleton , G. Michael Schneider
ACM Computing Surveys (CSUR) September 1975

Volume 7 Issue 3

5 An admission control scheme for predictable server response time for web accesses 98%
Xiangping Chen , Prasant Mohapatra , Huamin Chen
Proceedings of the tenth international conference on World Wide Web April 2001

6 Erlang capacity and uniform approximations for shared unbuffered resources 98%
Debasis Mitra , John A. Morrison
IEEE/ACM Transactions on Networking (TON) December 1994
Volume 2 Issue 6

7 Statistical multiplexing and mix-dependent alternative routing in multiservice VP networks 97%
Ching-Fong Su , Gustavo de Veciana
IEEE/ACM Transactions on Networking (TON) February 2000
Volume 8 Issue 1

8 Connection admission control for capacity-varying networks with stochastic capacity change times 97%
J. Siwko , I. Rubin
IEEE/ACM Transactions on Networking (TON) June 2001
Volume 9 Issue 3
Many connection-oriented networks, such as low Earth orbit satellite (LEOS) systems and networks providing multipriority service using advance reservations, have capacities which vary over time. Connection admission control (CAC) policies which only use current capacity information may lead to intolerable dropping of admitted connections whenever network capacity decreases. We present the admission limit curve (ALC) for capacity-varying networks with random capacity change times. We prove ...

9 The hierarchical simulation language HSL: a versatile tool for process-oriented simulation 97%
D. P. Sanderson , R. Sharma , R. Rozin , S. Treu
ACM Transactions on Modeling and Computer Simulation (TOMACS) April 1991
Volume 1 Issue 2

10 Microeconomics and the Market for Computer Services 96%
Ira W. Cotton
ACM Computing Surveys (CSUR) June 1975
Volume 7 Issue 2

11 Interactive modeling and simulation of transaction flow or network models using the ADA simulation support environment 96%
Heimo H. Adelsberger
Proceedings of the 15th conference on Winter Simulation - Volume 2 December 1983
The Ada Simulation Support Environment (ASSE) is a software system, with the

purpose to support the development and maintenance of simulation models written in Ada throughout their life cycle. We describe here the transaction flow or network part of the ASSE, which allows to build models like in GPSS or SLAM. Our view of such models is slightly different from that of the above mentioned languages, which is demonstrated in detail by the server/resource process. The design stresses modular to ...

12 Borrow, copy or steal?: loans and larceny in the orthodox canonical form 96%

Anthony J. H. Simons

ACM SIGPLAN Notices , Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications October 1998

Volume 33 Issue 10

Dynamic memory management in C++ is complex, especially across the boundaries of library abstract data types. C++ libraries designed in the orthodox canonical form (OCF) alleviate some of the problems by ensuring that classes which manage any kind of heap structures faithfully copy and delete these. However, in certain common circumstances, OCF heap structures are wastefully copied multiple times. General reference counting is not an option in OCF, since a shared body violates the intended value ...

13 Optimal pricing for multiple services in telecommunications networks offering quality-of-service guarantees 96%

Neil J. Keon , G. Anandalingam

IEEE/ACM Transactions on Networking (TON) February 2003

Volume 11 Issue 1

We consider pricing for multiple services offered over a single telecommunications network. Each service has quality-of-service (QoS) requirements that are guaranteed to users. Service classes may be defined by the type of service, such as voice, video, or data, as well as the origin and destination of the connection provided to the user. We formulate the optimal pricing problem as a nonlinear integer expected revenue optimization problem. We simultaneously solve for prices and the resource allo ...

14 Integrated networks that overflow speech and data between component networks 96%

Susan Lincke-Salecker , Cynthia S. Hood

International Journal of Network Management July 2002

Volume 12 Issue 4

As cellular networks diversify by expanding the number of services, cell sizes, and generations of technologies supported, it becomes possible to overflow dual-mode terminals using vertical handovers to other cellular networks or 'component networks'. This study varies call placement algorithms, by defining, modeling, and evaluating Overflow and Return policies.

15 Performance Workload Char. and Adaptation: Aliasing on the world wide web: prevalence and performance implications 95%

Terence Kelly , Jeffrey Mogul

Proceedings of the eleventh international conference on W rld Wide Web May 2002

Aliasing occurs in Web transactions when requests containing different URLs elicit replies containing identical data payloads. Conventional caches associate stored data with URLs and can therefore suffer redundant payload transfers due to aliasing and

other causes. Existing research literature, however, says little about the prevalence of aliasing in user-initiated transactions, or about redundant payload transfers in conventional Web cache hierarchies.This paper quantifies the extent of aliasin ...

16 Expressing and enforcing distributed resource sharing agreements 95%

Tao Zhao , Vijay Karamcheti

Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM) November 2000

Advances in computing and networking technology, and an explosion in information sources has resulted in a growing number of distributed systems being constructed out of resources contributed by multiple sources. Use of such resources is typically governed by sharing agreements between owning principals, which limit both who can access a resource and in what quantity. Despite their increasing importance, existing resource management infrastructures offer only limited support for the expres ...

17 Capacity Bounds for Multiresource Queues 95%

Kenneth J. Omahen

Journal of the ACM (JACM) October 1977

Volume 24 Issue 4

18 Manufacturing applications: Computer-aided manufacturing simulation (CAMS) generation for interactive analysis: concepts, techniques, and issues 95%

Boonserm Kulvatunyou , Richard A. Wysk

Proceedings of the 33nd conference on Winter simulation December 2001

Simulation model is usually developed as a one-time use analytical model by a system analyst (usually from external firm) rather than for a routine and interactive use by a shop floor engineer. This is because it usually takes longer time to generate a result from the simulation, and the simulation model of manufacturing system is usually too sophisticated and time-consuming to use as an interactive tool by the manufacturing/production engineer. A CAMS reduces this complication by encapsulating ...

19 Data-driven simulation of networks with manufacturing blocking 95%

Gordon M. Clark , Charles R. Cash

Proceedings of the 25th conference on Winter simulation December 1993

20 Efficient decomposition methods for the analysis of multi-facility blocking models 94%

Adrian E. Conway , Eugene Pinsky , Srinivasan Tridandapani

Journal of the ACM (JACM) July 1994

Volume 41 Issue 4

Three new decomposition methods are developed for the exact analysis of stochastic multi-facility blocking models of the product-form type. The first is a basic decomposition algorithm that reduces the analysis of blocking probabilities to that of two separate subsystems. The second is a generalized M-subsystem decomposition method. The third is a more elaborate and efficient incremental decomposition technique. All of the algorithms exploit the sparsity of locality that can be found in the ...

Results 1 - 20 f 200 **sh rt listing**

Prev Page 1 2 3 4 5 6 7 8 9 10 Next Page

h c g e cf